Lecture 1:

Why Parallelism?

Why Efficiency?

Parallel Computing
Stanford CS149, Fall 2023

One common definition

A parallel computer is a collection of processing elements that cooperate to solve problems quickly

We care about performance and we care about efficiency

We’re going to use multiple processors to get it

DEMO 1
(CS149 Fall 2023's first parallel program)

Speedup

One major motivation of using parallel processing: achieve a speedup

For a given problem:

$$\text{speedup(using P processors)} = \frac{\text{execution time (using 1 processor)}}{\text{execution time (using P processors)}}$$

Class observations from demo 1

- **Communication limited the maximum speedup achieved**
  - In the demo, the communication was telling each other the partial sums

- **Minimizing the cost of communication improved speedup**
  - Moved students ("processors") closer together (or let them shout)

DEMO 2

(scaling up to four “processors”)

Class observations from demo 2

- Imbalance in work assignment limited speedup
  - Some students ("processors") ran out work to do (went idle), while others were still working on their assigned task

- Improving the distribution of work improved speedup

DEMO 3

(massively parallel execution)

Class observations from demo 3

- The problem I just gave you has a significant amount of communication compared to computation

- Communication costs can dominate a parallel computation, severely limiting speedup

Course theme 1:

Designing and writing parallel programs ... <u>that scale</u>!

- **Parallel thinking**
  1. **Decomposing work into pieces that can safely be performed in parallel**
  2. **Assigning work to processors**
  3. **Managing communication/synchronization between the processors so that it does not limit speedup**

- **Abstractions/mechanisms for performing the above tasks**
  - **Writing code in popular parallel programming languages**

Course theme 2:

Parallel computer hardware implementation: how parallel computers work

- Mechanisms used to implement abstractions efficiently
  - Performance characteristics of implementations
  - Design trade-offs: performance vs. convenience vs. cost

- Why do I need to know about hardware?
  - Because the characteristics of the machine really matter (recall speed of communication issues in earlier demos)
  - Because you care about efficiency and performance (you are writing parallel programs after all!)

Course theme 3:

Thinking about efficiency

- **FAST != EFFICIENT**

- **Just because your program runs faster on a parallel computer, it does not mean it is using the hardware efficiently**
  - **Is 2x speedup on computer with 10 processors a good result?**

- **Programmer's perspective: make use of provided machine capabilities**

- **HW designer's perspective: choosing the right capabilities to put in system (performance/cost, cost = silicon area?, power?, etc.)**

Why parallelism?

Some historical context: why avoid parallel processing?

- Single-threaded CPU performance doubling ~ every 18 months
- Implication: working to parallelize your code was often not worth the time
  - Software developer does nothing, code gets faster next year. Woot!

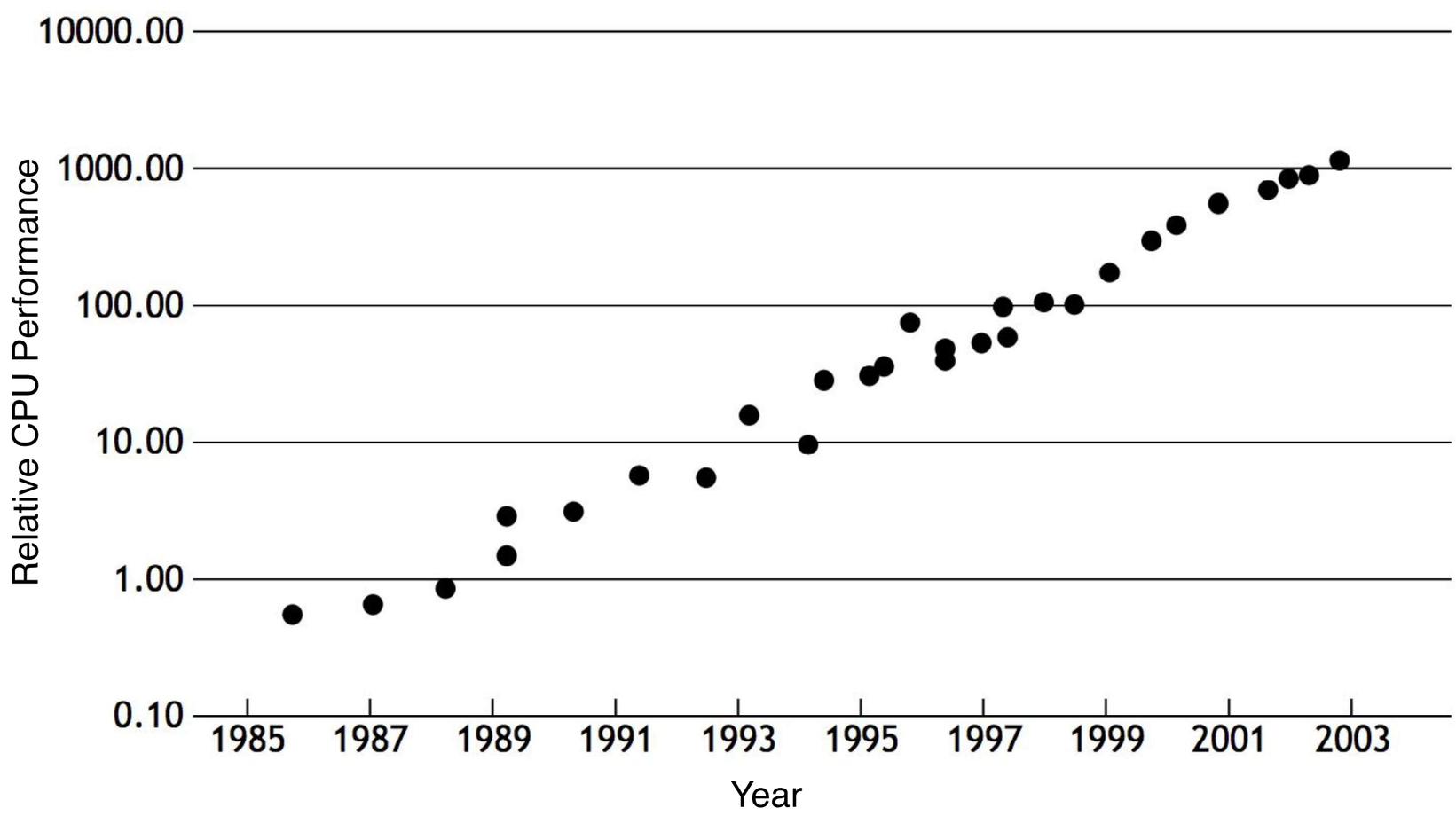


Image credit: Olukutun and Hammond, ACM Queue 2005

Until ~15 years ago: two significant reasons for processor performance improvement

1. Exploiting instruction-level parallelism (superscalar execution)
2. Increasing CPU clock frequency

What is a computer program?

Here is a program written in C

```
int main(int argc, char** argv) {

    int x = 1;

    for (int i=0; i<10; i++) {
        x = x + x;
    }

    printf("%d\n", x);

    return 0;
}
```

What is a program? (from a processor's perspective)

A program is just a list of processor instructions!

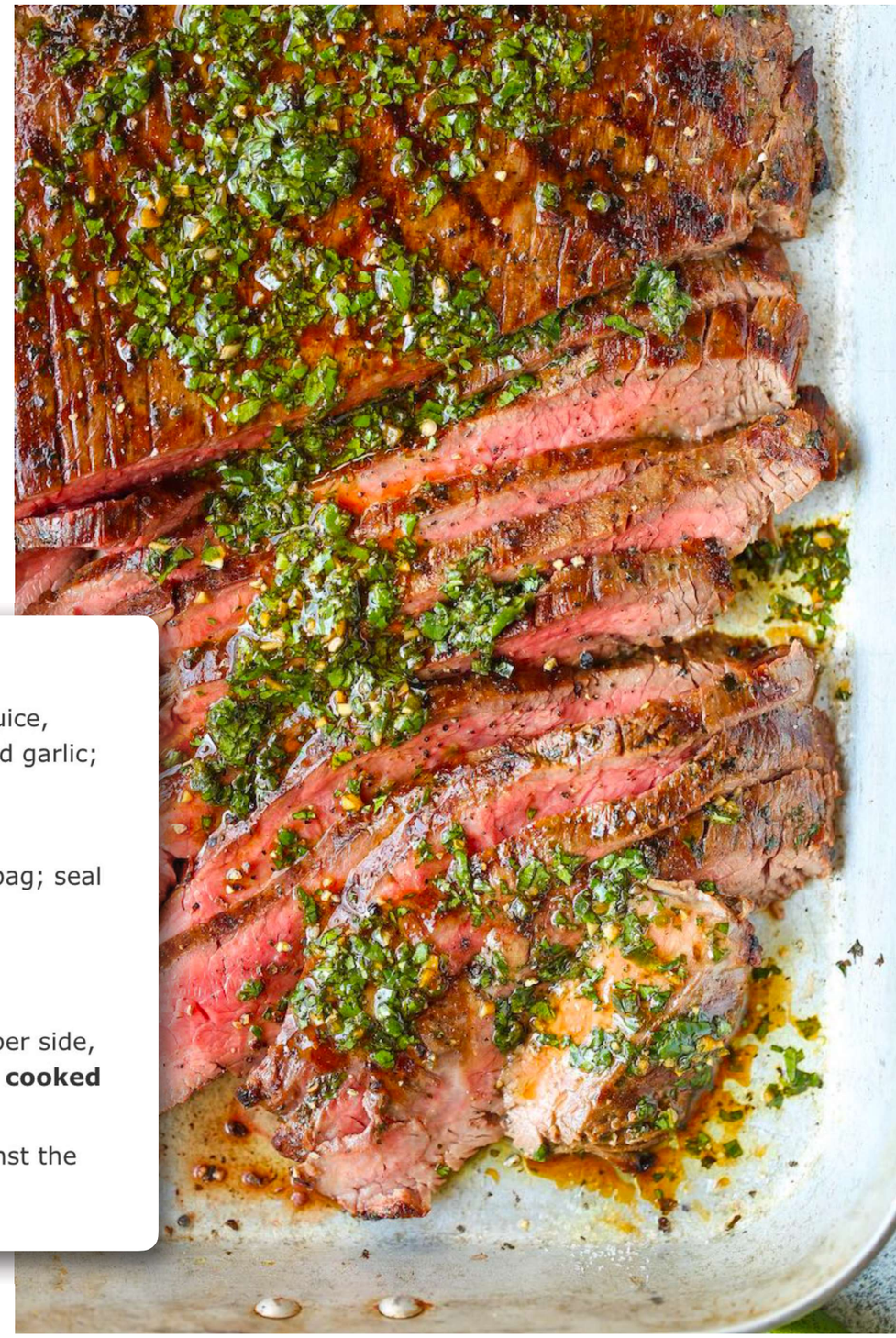
```
int main(int argc, char** argv) {

  int x = 1;

  for (int i=0; i<10; i++) {
    x = x + x;
  }

  printf("%d\n", x);

  return 0;
}
```



```
_main:
100000f10:    pushq    %rbp
100000f11:    movq %rsp, %rbp
100000f14:    subq $32, %rsp
100000f18:    movl $0, -4(%rbp)
100000f1f:    movl %edi, -8(%rbp)
100000f22:    movq %rsi, -16(%rbp)
100000f26:    movl $1, -20(%rbp)
100000f2d:    movl $0, -24(%rbp)
100000f34:    cmpl $10, -24(%rbp)
100000f38:    jge  23 <_main+0x45>
100000f3e:    movl -20(%rbp), %eax
100000f41:    addl -20(%rbp), %eax
100000f44:    movl %eax, -20(%rbp)
100000f47:    movl -24(%rbp), %eax
100000f4a:    addl $1, %eax
100000f4d:    movl %eax, -24(%rbp)
100000f50:    jmp  -33 <_main+0x24>
100000f55:    leaq 58(%rip), %rdi
100000f5c:    movl -20(%rbp), %esi
100000f5f:    movb $0, %al
100000f61:    callq    14
100000f66:    xorl %esi, %esi
100000f68:    movl %eax, -28(%rbp)
100000f6b:    movl %esi, %eax
100000f6d:    addq $32, %rsp
100000f71:    popq %rbp
100000f72:    rets
```

Kind of like the instructions in a recipe for your favorite meals

Mmm, carne asada

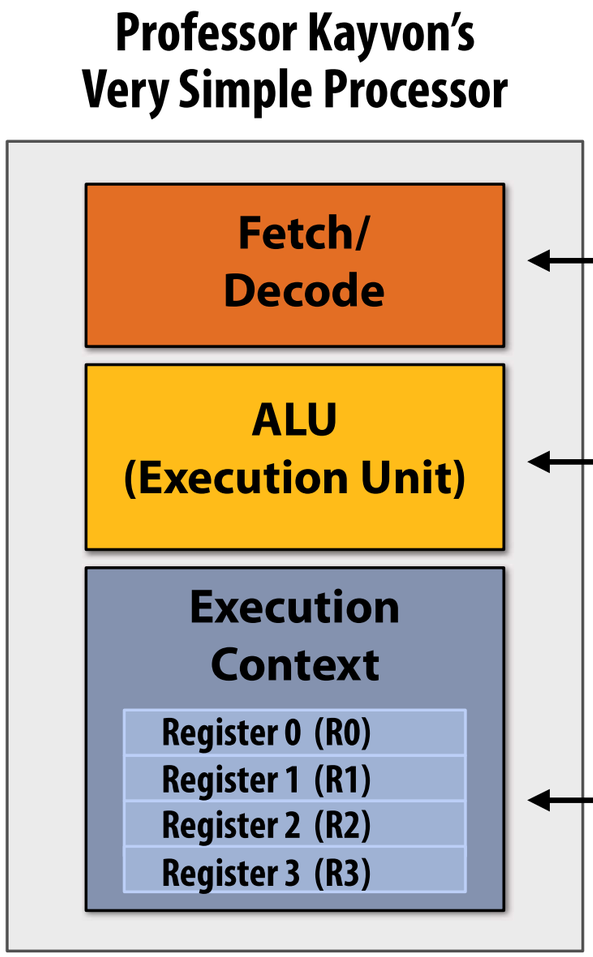


Instructions

1. In a large mixing bowl combine orange juice, olive oil, cilantro, lime juice, lemon juice, white wine vinegar, cumin, salt and pepper, jalapeno, and garlic; whisk until well combined.
2. Reserve ⅓ cup of the marinade; cover the rest and refrigerate.
3. Combine remaining marinade and steak in a large resealable freezer bag; seal and refrigerate for at least 2 hours, or overnight.
4. Preheat grill to HIGH heat.
5. Remove steak from marinade and lightly pat dry with paper towels.
6. Add steak to the preheated grill and cook for another 6 to 8 minutes per side, or until desired doneness. **Note that flank steak tastes best when cooked to rare or medium rare because it's a lean cut of steak.**
7. Remove from heat and let rest for 10 minutes. Thinly slice steak against the grain, garnish with reserved cilantro mixture, and serve.

What does a processor do?

A processor executes instructions

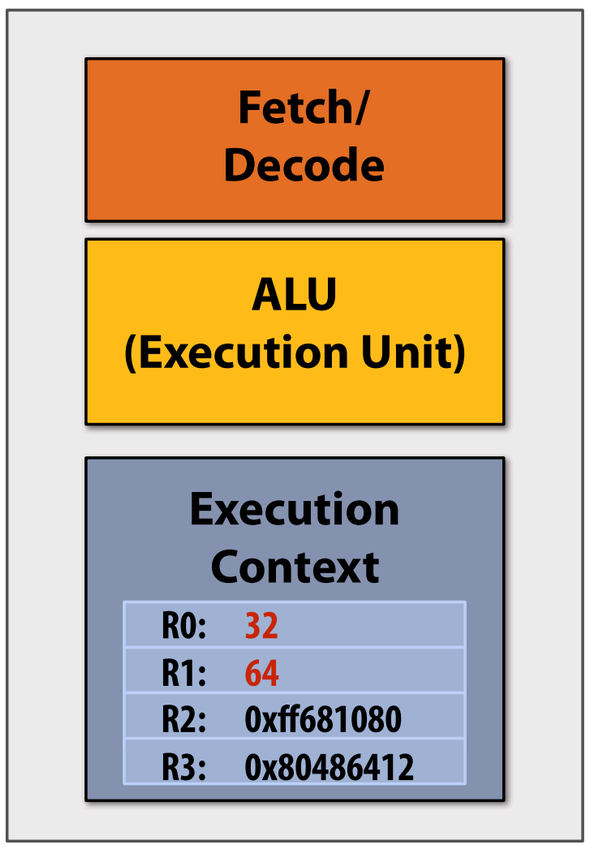


Determine what instruction to run next

Execution unit: performs the operation described by an instruction, which may modify values in the processor's registers or the computer's memory

Registers: maintain program state: store value of variables used as inputs and outputs to operations

One example instruction: add two numbers



Step 1:
Processor gets next program instruction from memory (figure out what the processor should do next)

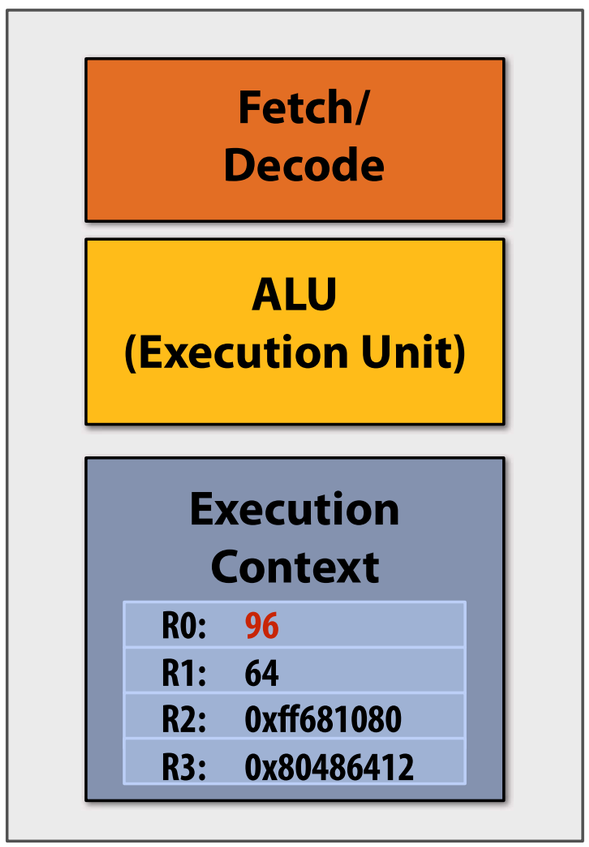
`add R0 ← R0, R1`

"Please add the contents of register R0 to the contents of register R1 and put the result of the addition into register R0"

Step 2:
Get operation inputs from registers
Contents of R0 input to execution unit: 32
Contents of R1 input to execution unit: 64

Step 3:
Perform addition operation:
Execution unit performs arithmetic, the result is: 96

One example instruction: add two numbers



Step 1:
Processor gets next program instruction from memory
(figure out what the processor should do next)

`add R0 ← R0, R1`

"Please add the contents of register R0 to the contents of register R1 and put the result of the addition into register R0"

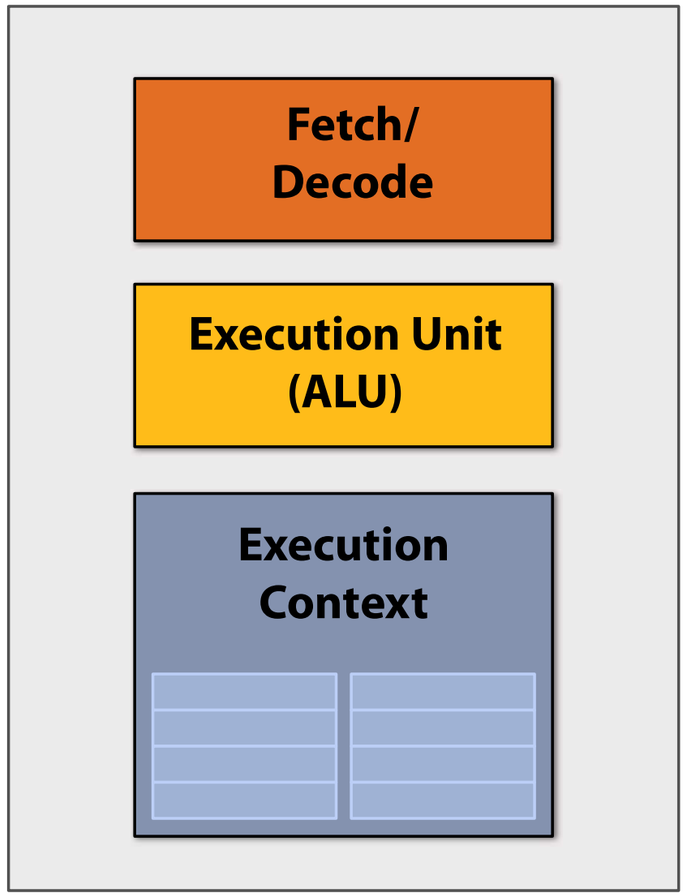
Step 2:
Get operation inputs from registers
Contents of R0 input to execution unit: 32
Contents of R1 input to execution unit: 64

Step 3:
Perform addition operation:
Execution unit performs arithmetic, the result is: 96

Step 4:
Store result 96 back to register R0

Execute program

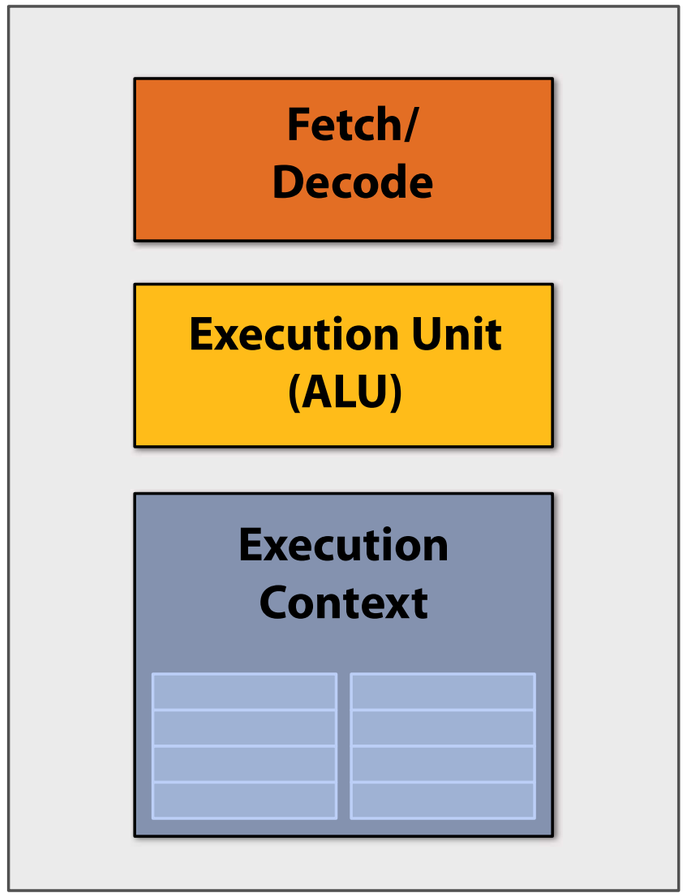
My very simple processor: executes one instruction per clock




```
ld   r0, addr[r1]
mul  r1, r0, r0
mul  r1, r1, r0
...
...
...
...
...
...
st   addr[r2], r0
```

Execute program

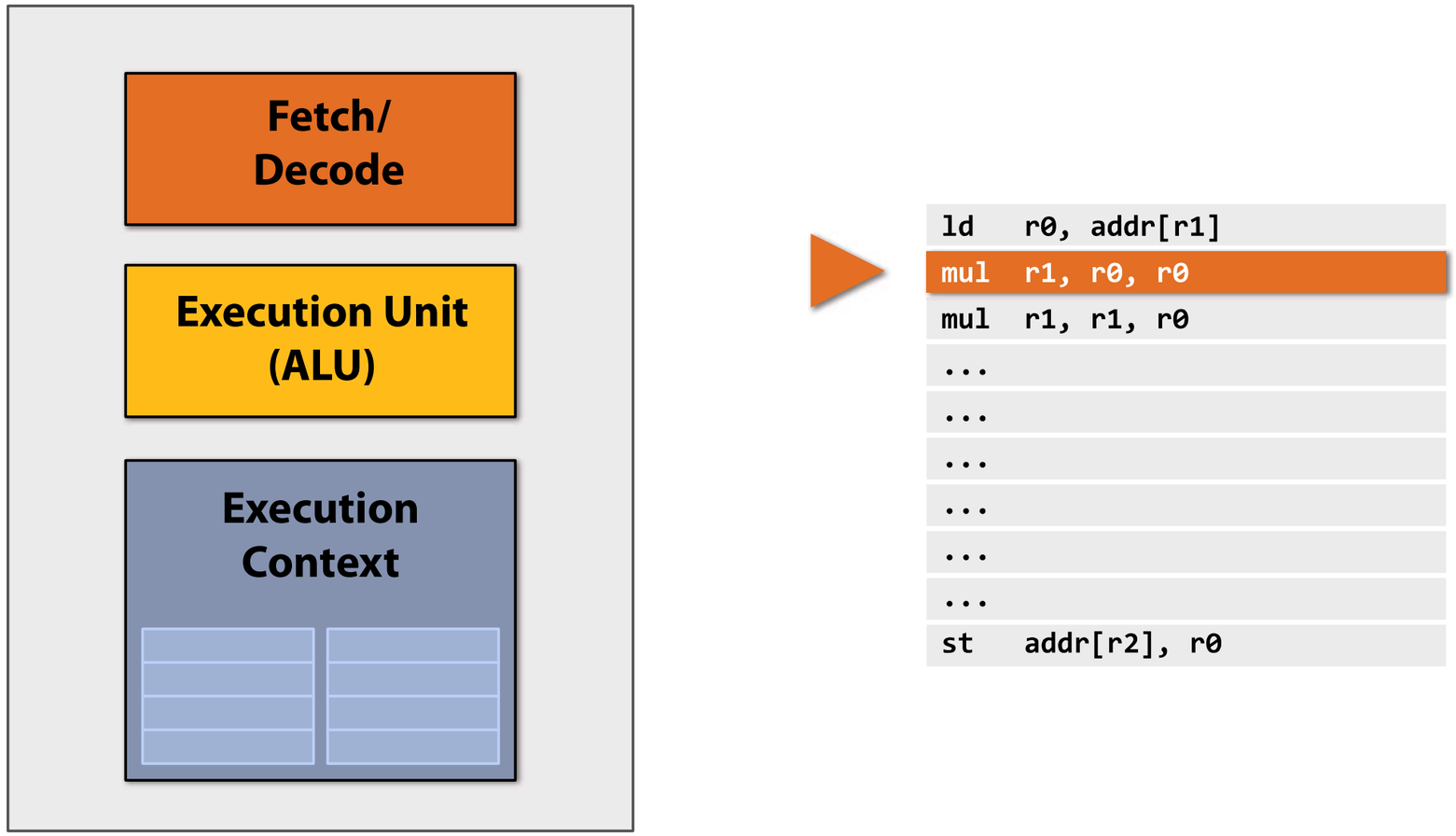
My very simple processor: executes one instruction per clock




```
ld   r0, addr[r1]
mul  r1, r0, r0
mul  r1, r1, r0
...
...
...
...
...
...
st   addr[r2], r0
```

Execute program

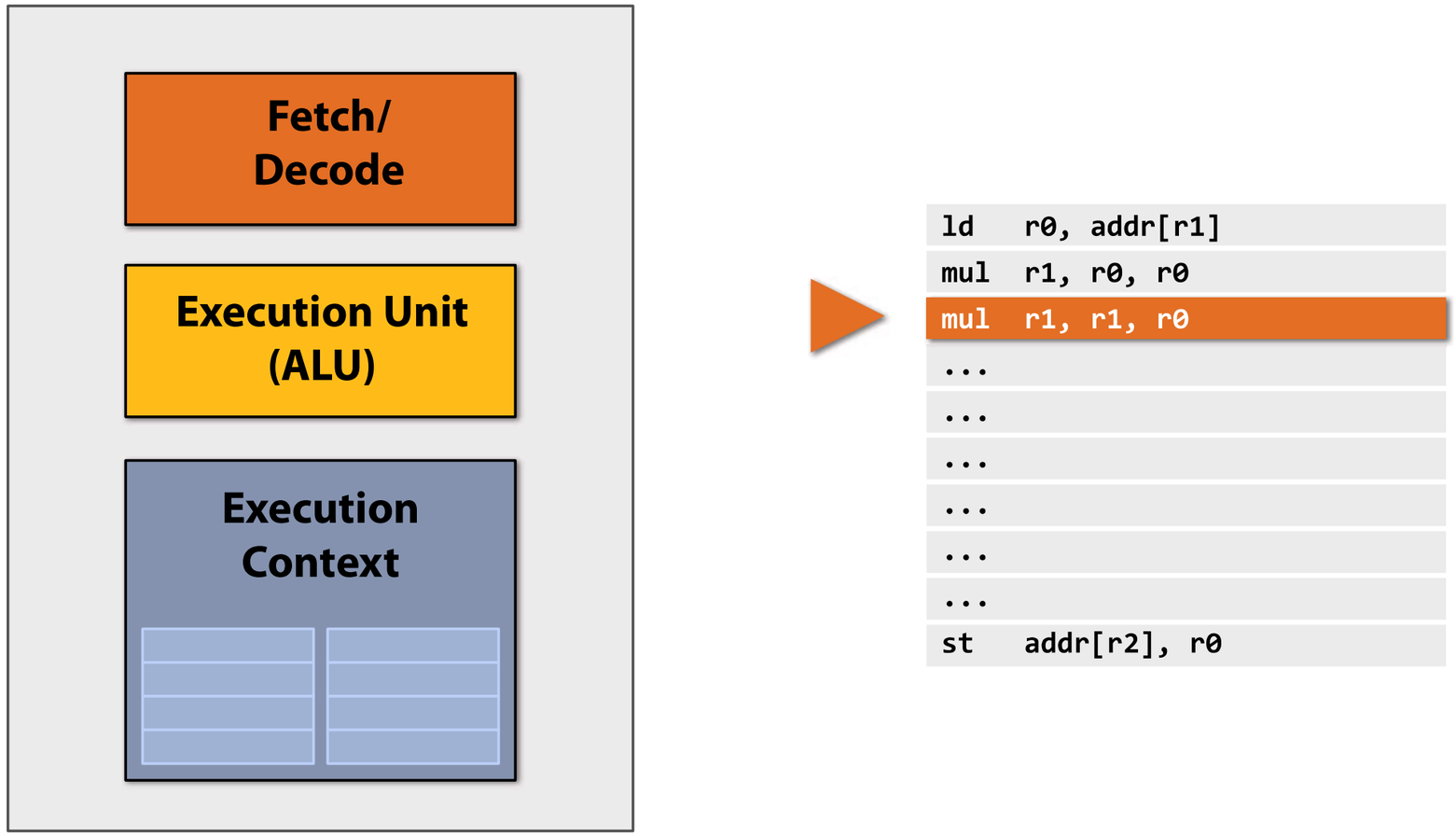
My very simple processor: executes one instruction per clock




```
ld   r0, addr[r1]
mul  r1, r0, r0
mul  r1, r1, r0
...
...
...
...
...
st   addr[r2], r0
```

Execute program

My very simple processor: executes one instruction per clock




```
ld   r0, addr[r1]
mul  r1, r0, r0
mul  r1, r1, r0
...
...
...
...
...
...
st   addr[r2], r0
```

Review of how computers work…

What is a computer program? (from a processor's perspective)

It is a list of instructions to execute!

What is an instruction?

It describes an operation for a processor to perform.

Executing an instruction typically modifies the computer's state.

What do I mean when I talk about a computer's "state"?

The values of program data, which are stored in a processor's registers or in memory.

Lets consider a very simple piece of code

```
a = x*x + y*y + z*z
```

Consider the following five instruction program:

Assume register R0 = x, R1 = y, R2 = z

```
1 mul R0, R0, R0
2 mul R1, R1, R1
3 mul R2, R2, R2
4 add R0, R0, R1
5 add R3, R0, R2
```

R3 now stores value of program variable 'a'

This program has five instructions, so it will take five clocks to execute, correct?

Can we do better?

What if up to two instructions can be performed at once?

```
a = x*x + y*y + z*z
```

Assume register
R0 = x, R1 = y, R2 = z

```
1 mul R0, R0, R0
2 mul R1, R1, R1
3 mul R2, R2, R2
4 add R0, R0, R1
5 add R3, R0, R2
```

R3 now stores value of program variable 'a'